



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/037,040	12/21/2001	Betty A. McDaniel	Baker 4-15-3-2-2	5546
7590 03/30/2005				
John L. DeAngelis, Jr., Esquire Holland & Knight LLP 1499 S. Harbor City Blvd., Suite 201 Melbourne, FL 32901		EXAMINER BELL, MELTIN		
		ART UNIT PAPER NUMBER 2129		

DATE MAILED: 03/30/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

**RECEIVED**

**APR 12 2005**

**Technology Center 2100**

<b>Office Action Summary</b>	<b>Application No.</b> 10/037,040	<b>Applicant(s)</b> MCDANIEL ET AL.	
	<b>Examiner</b> Meltin Bell	<b>Art Unit</b> 2121	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

#### Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

#### Status

- 1) ☒ Responsive to communication(s) filed on 21 December 2001 and 29 August 2003.
- 2a) ☐ This action is FINAL. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

#### Disposition of Claims

- 4) ☒ Claim(s) 1-25 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☒ Claim(s) 1-25 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

#### Application Papers

- 9) ☒ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 12 April 2002 is/are: a) ☐ accepted or b) ☒ objected to by the Examiner.  
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).  
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

#### Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).  
a) ☐ All b) ☐ Some \* c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
  - ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
  - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- \* See the attached detailed Office action for a list of the certified copies not received.

#### Attachment(s)

- |   |   |
|---|---|
| 1) <input checked="" type="checkbox"/> Notice of References Cited (PTO-892)   | 4) <input type="checkbox"/> Interview Summary (PTO-413)<br>Paper No(s)/Mail Date. _____ |
| 2) <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948)  | 5) <input type="checkbox"/> Notice of Informal Patent Application (PTO-152)             |
| 3) <input checked="" type="checkbox"/> Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)<br>Paper No(s)/Mail Date <u>8/29/03; 4/16/02</u> . | 6) <input type="checkbox"/> Other: _____  |

### **DETAILED ACTION**

This action is responsive to application **10/037,040** filed **12/21/2001**.

Claims 1-25 have been examined.

#### ***Drawings***

The drawings have not been checked to the extent necessary to determine the presence of all possible minor errors. Applicant's cooperation is required in correcting any errors of which applicant may become aware in the drawings.

The drawings are objected to because:

- Fig. 4, item 74 would read well labeled PROCESSING ENGINE as suggested on page 8, line 27

A proposed drawing correction or corrected drawings are required in reply to the Office action to avoid abandonment of the application. The objection to the drawings will not be held in abeyance.

#### ***Specification***

The specification has not been checked to the extent necessary to determine the presence of all possible minor errors. Applicant's cooperation is required in correcting any errors of which applicant may become aware in the specification.

The disclosure is objected to because of the following informalities:

Art Unit: 2121

- The discussion of Fig. 1 in the Background of the Invention section page 3, line 17 through page 5, line 13 would read well placed in the Detailed Description of the Invention section beginning on page 7
- '60' on page 10, line 9-11 would read well as '80'
- '64' on page 10, line 11 would read well as '84'

Appropriate correction is required.

### ***Claim Objections***

Claim 7 and 23 are objected to because of the following informalities:

#### **Regarding claim 7:**

- 'wherein a' on page 13, line 6 would read well as 'wherein the decision tree structure comprises a'
- 'of the plurality' on page 13, line 7 would read well as 'of a plurality'

#### **Regarding claim 23:**

- 'method' on page 15, line 19 would read well as 'apparatus'

Appropriate correction is required.

### ***Claim Rejections - 35 USC § 101***

35 U.S.C. 101 reads as follows:

Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

Claims 1-3, 5-19 and 21-23 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. The language of the claims (e.g. "search object", "entry", "knowledge base") raise a question as to whether the claims are directed merely to an abstract idea that is not tied to a technological art, environment or machine which would result in a practical application producing a concrete, useful, and tangible result to form the basis of statutory subject matter under 35 U.S.C. 101. For example, if the independent claims were amended to recite a computer-implemented method or apparatus and required performance of a result outside of a computer, it will be statutory in most cases since use of technology permits the function of the descriptive material to be realized.

***Claim Rejections - 35 USC § 103***

To expedite a complete examination of the instant application, the claims rejected under 35 U.S.C. 101 (nonstatutory) above are further rejected as set forth below in anticipation of applicant amending these claims to place them within the four statutory categories of invention.

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

This application currently names joint inventors. In considering patentability of the claims under 35 U.S.C. 103(a), the Office presumes that the subject matter of the various claims was commonly owned at the time any inventions covered therein were made absent any evidence to the contrary. Applicant is advised of the obligation under 37 CFR 1.56 to point out the inventor and invention dates of each claim that was not commonly owned at the time a later invention was made in order for the Office to consider the applicability of 35 U.S.C. 103(c) and potential 35 U.S.C. 102(e), (f) or (g) prior art under 35 U.S.C. 103(a).

Claims 1-3, 6, 13, 15, 18-19 and 22 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bennett* United States Patent Number (USPN) 5,813,001 "Method for performing optimized intelligent searches of knowledge bases using submaps associated with search objects" (Sep. 22, 1998) in view of *Bialkowski et al* USPN 5,463,777 "System for segmenting data packets to form binary decision trees which determine filter masks combined to filter the packets for forwarding" (Oct. 31, 1995).

**Regarding claim 1:**

*Bennett* teaches,

- A method for determining whether a search object matches an entry in a knowledge base (Abstract), wherein the knowledge base comprises a plurality of search nodes, and a plurality of links joining two search nodes (Fig. 2; column 6, lines 45-63), said method comprising
  - reading (column 4, lines 66-67; column 5, lines 1-22) a first search node from the first memory

Art Unit: 2121

- comparing the first search node with at least a portion of the search object (column 5, lines 23-45)

- based on the comparing step, traversing a search path (column 8, lines 59-61) from the first search node to a second search node via the joining link

However, *Bennett* doesn't explicitly teach the knowledge base comprises a decision tree structure while *Bialkowski et al* teaches,

- A method (Abstract; column 1, lines 62-67), wherein the knowledge base comprises a decision tree structure comprising a plurality of search nodes, and a plurality of links joining two search nodes (column 3, lines 40-52), said method comprising

- storing a first portion of the decision tree structure in a first memory (Fig. 1; column 2, lines 9-22), wherein the first portion comprises a first plurality of search nodes and interconnecting links

- storing a second portion of the decision tree structure in a second memory (Fig. 1; column 6, lines 22-41), wherein the second portion comprises a second plurality of search nodes and interconnecting links

Motivation - The portions of the claimed method would have been a highly desirable feature in this art for maintaining memory requirements and other hardware needs at a minimum (*Bialkowski et al*, column 1, lines 32-39). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made, to modify *Bennett* as taught by *Bialkowski et al* for the purpose of maintaining memory requirements.

**Regarding claim 2:**

The rejection of claim 2 is similar to that for claim 1 as recited above since the stated limitations of the claim are set forth in the references. Claim 2's limitations difference is taught in *Bennett*:

- reading the second search node and comparing at least a portion of the search object with the second search node (column 5, lines 7-45)

**Regarding claim 3:**

The rejection of claim 3 is similar to that for claim 1 as recited above since the stated limitations of the claim are set forth in the references. Claim 3's limitations difference is taught in *Bennett*:

- the steps of reading, comparing and traversing are repeated until the second portion of the decision tree is traversed to an end thereof (column 16, lines 35-53)

**Regarding claim 6:**

The rejection of claim 6 is the same as that for claim 1 as recited above since the stated limitations of the claim are set forth in the references.

**Regarding claim 13:**

The rejection of claim 13 is the same as that for claims 1 and 3 as recited above since the stated limitations of the claim are set forth in the references.

**Regarding claim 15:**

The rejection of claim 15 is similar to that for claim 1 as recited above since the stated limitations of the claim are set forth in the references. Claim 15's limitations difference is taught in *Bialkowski et al*:



Art Unit: 2121

- the decision tree structure comprises a plurality of contiguous (column 1, line 67; column 2, lines 1-8) tree levels, wherein each tree level further comprises a search node and link to a search node of the next adjacent tree level

**Regarding claim 18:**

*Bennett* teaches,

- An apparatus for determining whether a search object matches any entry in a knowledge base (Abstract), wherein the knowledge base comprises a plurality of links between adjacent search nodes (Fig. 2; column 6, lines 45-63)

However, *Bennett* doesn't explicitly teach the knowledge base comprises a decision tree structure while *Bialkowski et al* teaches,

- An apparatus (Abstract; column 1, lines 62-67) wherein the knowledge base comprises a decision tree structure comprising a plurality of links between adjacent search nodes (column 3, lines 40-52), said apparatus comprising
  - a first memory storing a first portion of the decision tree structure (Fig. 1; column 2, lines 9-22)
  - a second memory storing a second portion of the decision tree structure (Fig. 1; column 6, lines 22-41)
  - a processor (column 1, lines 22-28) for matching at least a portion of the search object with a search node and for traversing through the decision tree structure in response to the match

Motivation - The portions of the claimed apparatus would have been a highly desirable feature in this art for maintaining memory requirements and other hardware needs at a

Art Unit: 2121

minimum (*Bialkowski et al*, column 1, lines 32-39). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made, to modify *Bennett* as taught by *Bialkowski et al* for the purpose of maintaining memory requirements.

**Regarding claim 19:**

The rejection of claim 19 is the same as that for claims 18 and 1 as recited above since the stated limitations of the claim are set forth in the references.

**Regarding claim 22:**

The rejection of claim 22 is the same as that for claims 18 and 1 as recited above since the stated limitations of the claim are set forth in the references.

Claims 4 and 20 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bennett* in view of *Bialkowski et al* and in further view of *Pollack et al* USPN 6,571,238 "System for regulating flow of information to user by using time dependent function to adjust relevancy threshold" (Filed Jun. 11, 1999).

**Regarding claim 4:**

*Bennett* teaches,

- A method for determining whether a search object matches) an entry in a knowledge base (Abstract), wherein the knowledge base comprises a plurality of search nodes, and a plurality of links joining two search nodes (Fig. 2; column 6, lines 45-63), said method comprising

Art Unit: 2121

- reading (column 4, lines 66-67; column 5, lines 1-22) a first search node from the first memory

- comparing the first search node with at least a portion of the search object (column 5, lines 23-45)

- based on the comparing step, traversing a search path (column 8, lines 59-61) from the first search node to a second search node via the joining link

However, *Bennett* doesn't explicitly teach the knowledge base comprises a decision tree structure or the step of reading is executed by a processor formed in an integrated circuit, and wherein the first memory is formed on the integrated circuit, such that the step of reading search nodes from the first memory executes faster than the step of reading search nodes from the second memory while *Bialkowski et al* teaches,

- A method (Abstract; column 1, lines 62-67), wherein the knowledge base comprises a decision tree structure comprising a plurality of search nodes, and a plurality of links joining two search nodes (column 3, lines 40-52), said method comprising

- storing a first portion of the decision tree structure in a first memory (Fig. 1; column 2, lines 9-22), wherein the first portion comprises a first plurality of search nodes and interconnecting links

- storing a second portion of the decision tree structure in a second memory (Fig. 1; column 6, lines 22-41), wherein the second portion comprises a second plurality of search nodes and interconnecting links

*Pollack et al* teaches,

- the step of reading is executed by a processor formed in an integrated circuit, and

Art Unit: 2121

wherein the first memory is formed on the integrated circuit, such that the step of reading search nodes from the first memory executes faster than the step of reading search nodes from the second memory (column 10, lines 54-67; column 11, lines 1-3)

Motivation - The portions of the claimed method would have been a highly desirable feature in this art for maintaining memory requirements and other hardware needs at a minimum (*Bialkowski et al*, column 1, lines 32-39) and managing data movement (*Pollack et al*, column 11, lines 6-10). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made, to modify *Bennett* as taught by *Bialkowski et al* and *Pollack et al* for the purpose of maintaining memory requirements and managing data movement.

**Regarding claim 20:**

The rejection of claim 20 is the same as that for claims 18 and 4 as recited above since the stated limitations of the claim are set forth in the references.

Claims 5 and 17 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bennett* in view of *Bialkowski et al* and in further view of *Nakano et al* USPN 6,636,802 "Data structure of digital map file" (PCT Filed Nov. 24, 1999).

**Regarding claim 5:**

*Bennett* teaches,

- A method for determining whether a search object matches) an entry in a knowledge base (Abstract), wherein the knowledge base comprises a plurality of search nodes,

Art Unit: 2121

and a plurality of links joining two search nodes (Fig. 2; column 6, lines 45-63), said

method comprising

- reading (column 4, lines 66-67; column 5, lines 1-22) a first search node from the first memory

- comparing the first search node with at least a portion of the search object (column 5, lines 23-45)

- based on the comparing step, traversing a search path (column 8, lines 59-61) from the first search node to a second search node via the joining link

However, *Bennett* doesn't explicitly teach the knowledge base comprises a decision tree structure or the first portion of the decision tree structure comprises the search nodes near the first search entry while *Bialkowski et al* teaches,

- A method (Abstract; column 1, lines 62-67), wherein the knowledge base comprises a decision tree structure comprising a plurality of search nodes, and a plurality of links joining two search nodes (column 3, lines 40-52), said method comprising

- storing a first portion of the decision tree structure in a first memory (Fig. 1; column 2, lines 9-22), wherein the first portion comprises a first plurality of search nodes and interconnecting links

- storing a second portion of the decision tree structure in a second memory (Fig. 1; column 6, lines 22-41), wherein the second portion comprises a second plurality of search nodes and interconnecting links

*Nakano et al* teaches,

- the first portion of the decision tree structure comprises the search nodes near the first search entry (column 41, lines 45-60)

Motivation - The portions of the claimed method would have been a highly desirable feature in this art for maintaining memory requirements and other hardware needs at a minimum (*Bialkowski et al*, column 1, lines 32-39) and speeding up the entry node search (*Nakano et al*, column 41, lines 60-62). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made, to modify *Bennett* as taught by *Bialkowski et al* and *Nakano et al* for the purpose of maintaining memory requirements and speeding up the entry node search.

**Regarding claim 17:**

The rejection of claim 17 is the same as that for claims 15 and 5 as recited herein since the stated limitations of the claim are set forth in the references:

Claims 7-9, 16 and 21 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bennett* in view of *Bialkowski et al* and in further view of *Vahalia et al* USPN 6,625,591 "Very efficient in-memory representation of large file system directories" (Filed Sep. 29, 2000).

**Regarding claim 7:**

*Bennett* teaches,

- A method for determining whether a search object matches) an entry in a knowledge base (Abstract), wherein the knowledge base comprises a plurality of search nodes,

Art Unit: 2121

and a plurality of links joining two search nodes (Fig. 2; column 6, lines 45-63), said method comprising

- reading (column 4, lines 66-67; column 5, lines 1-22) a first search node from the first memory
- comparing the first search node with at least a portion of the search object (column 5, lines 23-45)
- based on the comparing step, traversing a search path (column 8, lines 59-61) from the first search node to a second search node via the joining link

However, *Bennett* doesn't explicitly teach the knowledge base comprises a decision tree structure or a predetermined number of lower levels of the plurality of levels are stored in the first memory, and wherein the remaining plurality of levels are stored in the second memory while *Bialkowski et al* teaches,

- A method (Abstract; column 1, lines 62-67), wherein the knowledge base comprises a decision tree structure comprising a plurality of search nodes, and a plurality of links joining two search nodes (column 3, lines 40-52), said method comprising
  - storing a first portion of the decision tree structure in a first memory (Fig. 1; column 2, lines 9-22), wherein the first portion comprises a first plurality of search nodes and interconnecting links
  - storing a second portion of the decision tree structure in a second memory (Fig. 1; column 6, lines 22-41), wherein the second portion comprises a second plurality of search nodes and interconnecting links

*Vahalia et al* teaches,

Art Unit: 2121

- a predetermined number of lower levels of the plurality of levels are stored in the first memory, and wherein the remaining plurality of levels are stored in the second memory (column 13, line 67; column 14, lines 1-23)

Motivation - The portions of the claimed method would have been a highly desirable feature in this art for maintaining memory requirements and other hardware needs at a minimum (*Bialkowski et al*, column 1, lines 32-39) and managing data movement (*Pollack et al*, column 11, lines 6-10) and accelerating a search (*Vahalia et al*, column 2, lines 47-49). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made, to modify *Bennett* as taught by *Bialkowski et al* and *Vahalia et al* for the purpose of maintaining memory requirements and accelerating a search.

**Regarding claim 8:**

The rejection of claim 8 is the same as that for claim 7 as recited above since the stated limitations of the claim are set forth in the references.

**Regarding claim 9:**

The rejection of claim 9 is the same as that for claims 1 and 7 as recited above since the stated limitations of the claim are set forth in the references.

**Regarding claim 16:**

The rejection of claim 16 is the same as that for claims 15 and 7 as recited above since the stated limitations of the claim are set forth in the references.



**Regarding claim 21:**

The rejection of claim 21 is the same as that for claims 18 and 7 as recited above since the stated limitations of the claim are set forth in the references.

Claims 10-11 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bennett* in view of *Bialkowski et al* and in further view of *Friedberg* USPN 6,662,184 "Very efficient in-memory representation of large file system directories" (Filed Sep. 22, 2000).

**Regarding claim 10:**

*Bennett* teaches,

- A method for determining whether a search object matches) an entry in a knowledge base (Abstract), wherein the knowledge base comprises a plurality of search nodes, and a plurality of links joining two search nodes (Fig. 2; column 6, lines 45-63), said method comprising
  - reading (column 4, lines 66-67; column 5, lines 1-22) a first search node from the first memory
  - comparing the first search node with at least a portion of the search object (column 5, lines 23-45)
  - based on the comparing step, traversing a search path (column 8, lines 59-61) from the first search node to a second search node via the joining link

Art Unit: 2121

However, *Bennett* doesn't explicitly teach the knowledge base comprises a decision tree structure or the search object comprises a plurality of symbols while *Bialkowski et al* teaches,

- A method (Abstract; column 1, lines 62-67), wherein the knowledge base comprises a decision tree structure comprising a plurality of search nodes, and a plurality of links joining two search nodes (column 3, lines 40-52), said method comprising
  - storing a first portion of the decision tree structure in a first memory (Fig. 1; column 2, lines 9-22), wherein the first portion comprises a first plurality of search nodes and interconnecting links
  - storing a second portion of the decision tree structure in a second memory (Fig. 1; column 6, lines 22-41), wherein the second portion comprises a second plurality of search nodes and interconnecting links

*Friedberg* teaches,

- the search object comprises a plurality of symbols (column 10, lines 66-67; column 11, lines 1-10)

Motivation - The portions of the claimed method would have been a highly desirable feature in this art for maintaining memory requirements and other hardware needs at a minimum (*Bialkowski et al*, column 1, lines 32-39) and searching and retrieving data (*Friedberg*, Abstract). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made, to modify *Bennett* as taught by *Bialkowski et al* and *Friedberg* for the purpose of maintaining memory requirements and searching/retrieving data.

**Regarding claim 11:**

The rejection of claim 11 is the same as that for claim 10 as recited above since the stated limitations of the claim are set forth in the references.

Claim 12 is rejected under 35 U.S.C. 103(a) as being unpatentable over *Bennett* in view of *Bialkowski et al* and in further view of *Corl et al* USPN 6,772,223

"Configurable classification interface for networking devices supporting multiple action packet handling rules" (Filed Apr. 10, 2000).

**Regarding claim 12:**

*Bennett* teaches,

- A method for determining whether a search object matches) an entry in a knowledge base (Abstract), wherein the knowledge base comprises a plurality of search nodes, and a plurality of links joining two search nodes (Fig. 2; column 6, lines 45-63), said method comprising
  - reading (column 4, lines 66-67; column 5, lines 1-22) a first search node from the first memory
  - comparing the first search node with at least a portion of the search object (column 5, lines 23-45)
  - based on the comparing step, traversing a search path (column 8, lines 59-61) from the first search node to a second search node via the joining link

However, *Bennett* doesn't explicitly teach the knowledge base comprises a decision tree structure or the knowledge base comprises a classification engine of a

communications network processor for determining an attribute of the data input thereto, and wherein the second portion of the decision tree ends in a plurality of terminating nodes, the method further comprising repeating the steps of reading, comparing and traversing until a terminating node is reached, wherein the terminating node identifies the attribute of the input data while *Bialkowski et al* teaches,

- A method (Abstract; column 1, lines 62-67), wherein the knowledge base comprises a decision tree structure comprising a plurality of search nodes, and a plurality of links joining two search nodes (column 3, lines 40-52), said method comprising
- storing a first portion of the decision tree structure in a first memory (Fig. 1; column 2, lines 9-22), wherein the first portion comprises a first plurality of search nodes and interconnecting links
- storing a second portion of the decision tree structure in a second memory (Fig. 1; column 6, lines 22-41), wherein the second portion comprises a second plurality of search nodes and interconnecting links

*Corl et al* teaches,

- the knowledge base comprises a classification engine of a communications network processor for determining an attribute of the data input thereto (column 2, lines 11-27), and wherein the second portion of the decision tree ends in a plurality of terminating nodes, the method further comprising repeating the steps of reading, comparing and traversing until a terminating node is reached, wherein the terminating node identifies the attribute of the input data (column 2, lines 58-67; column 3 lines 1-28)

Art Unit: 2121

Motivation - The portions of the claimed method would have been a highly desirable feature in this art for maintaining memory requirements and other hardware needs at a minimum (*Bialkowski et al*, column 1, lines 32-39) and defining the types of actions that are to be applied to packets processed by a network processor device (*Corl et al*, Abstract). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made, to modify *Bennett* as taught by *Bialkowski et al* and *Corl et al* for the purpose of maintaining memory requirements and defining network processor packet action types.

Claims 14 and 23-25 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bennett* in view of *Bialkowski et al* and in further view of *Benayoun et al* USPN 6,516,319 B1 "Parallelized processing device for processing search keys based upon tree structure" (Filed May 11, 2000).

**Regarding claim 14:**

*Bennett* teaches,

- A method for determining whether a search object matches) an entry in a knowledge base (Abstract), wherein the knowledge base comprises a plurality of search nodes, and a plurality of links joining two search nodes (Fig. 2; column 6, lines 45-63), said method comprising
  - reading (column 4, lines 66-67; column 5, lines 1-22) a first search node from the first memory

- comparing the first search node with at least a portion of the search object (column 5, lines 23-45)

- based on the comparing step, traversing a search path (column 8, lines 59-61) from the first search node to a second search node via the joining link

However, *Bennett* doesn't explicitly teach the knowledge base comprises a decision tree structure or each one of the plurality of search nodes comprises an instruction and an address field, wherein the step of comparing further comprises comparing at least a portion of the search object with the instruction, and wherein the address field determines the second search node based on the comparing step while *Bialkowski et al* teaches,

- A method (Abstract; column 1, lines 62-67), wherein the knowledge base comprises a decision tree structure comprising a plurality of search nodes, and a plurality of links joining two search nodes (column 3, lines 40-52), said method comprising

- storing a first portion of the decision tree structure in a first memory (Fig. 1; column 2, lines 9-22), wherein the first portion comprises a first plurality of search nodes and interconnecting links

- storing a second portion of the decision tree structure in a second memory (Fig. 1; column 6, lines 22-41), wherein the second portion comprises a second plurality of search nodes and interconnecting links

*Benayoun et al* teaches,

- each one of the plurality of search nodes comprises an instruction and an address field, wherein the step of comparing further comprises comparing at least a portion of

Art Unit: 2121

the search object with the instruction, and wherein the address field determines the second search node based on the comparing step (column 8, lines 15-21)

Motivation - The portions of the claimed method would have been a highly desirable feature in this art for maintaining memory requirements and other hardware needs at a minimum (*Bialkowski et al*, column 1, lines 32-39) and searching for the tree leaf matching a search key (*Benayoun et al*, Abstract). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made, to modify *Bennett* as taught by *Bialkowski et al* and *Benayoun et al* for the purpose of maintaining memory requirements and matching a search key.

**Regarding claim 23:**

*Bennett* teaches,

- An apparatus for determining whether a search object matches any entry in a knowledge base (Abstract), wherein the knowledge base comprises a plurality of links connecting adjacent search nodes (Fig. 2; column 6, lines 45-63)

However, *Bennett* doesn't explicitly teach the knowledge base comprises a decision tree structure or first and second processors while *Bialkowski et al* teaches,

- An apparatus (Abstract; column 1, lines 62-67) wherein the knowledge base comprises a decision tree structure comprising a plurality of links connecting adjacent search nodes (column 3, lines 40-52), said **method** comprising

- a first memory storing a first portion of the decision tree structure (Fig. 1; column 2, lines 9-22)

- a second memory storing a second portion of the decision tree structure (Fig. 1; column 6, lines 22-41)

*Benayoun et al* teaches,

- a first processor (Fig. 1, item 30)
- a second processor (Fig. 1, item 32)
- wherein said first processor accesses said first memory, and wherein said second processor accesses said second memory for determining the search node that matches at least a portion of said search object (column 2, lines 20-46; column 9, lines 45-46)

Motivation - The portions of the claimed apparatus/method would have been a highly desirable feature in this art for maintaining memory requirements and other hardware needs at a minimum (*Bialkowski et al*, column 1, lines 32-39) and searching for the tree leaf matching a search key (*Benayoun et al*, Abstract). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made, to modify *Bennett* as taught by *Bialkowski et al* and *Benayoun et al* for the purpose of maintaining memory requirements and matching a search key.

**Regarding claim 24:**

The rejection of claim 24 is the same as that for claims 23 and 1 as recited above since the stated limitations of the claim are set forth in the references.

**Regarding claim 25:**

The rejection of claim 25 is similar to that for claim 23 as recited above since the stated limitations of the claim are set forth in the references. Claim 25's limitations difference is taught in *Benayoun et al*:



Art Unit: 2121

- the first processor and the second processor simultaneously execute tree searches for a plurality of search trees (column 8, lines 42-47)

### ***Conclusion***

The following prior art made of record is considered pertinent to applicant's disclosure:

- *van der Wal et al*; US 5963675 A; Pipelined pyramid processor for image processing systems
- *Nagral et al*; US 6260044 B1; Information storage and retrieval system for storing and retrieving the visual form of information from an application in a database
- *Srivastava et al*; US 6563952 B1; Method and apparatus for classification of high dimensional data
- *Tzeng*; US 6061712 A; Method for IP routing table look-up
- *Singh et al*; US 5983224 A; Method and apparatus for reducing the computational requirements of K-means data clustering
- *Lomet*; US 4611272 A; Key-accessed file organization
- *Wu et al*; US 6381607 B1; System of organizing catalog data for searching and retrieval
- *Israni et al*; US 5968109 A; System and method for use and storage of geographic data on physical media
- *Powers et al*; US 5404513 A; Method for building a database with multi-dimensional search tree nodes

- *Simonetti*; US 5295261 A; Hybrid database structure linking navigational fields having a hierarchial database structure to informational fields having a relational database structure
- *Zellweger*; US 5630125 A; Method and apparatus for information management using an open hierarchical data structure
- *Marquis*; US 5930805 A; Storage and retrieval of ordered sets of keys in a compact 0-complete tree
- *Demuynck et al*; Bmad-tree: an efficient data structure for parallel processing; Eighth IEEE Symposium on Parallel and Distributed Processing; 23-26 Oct. 1996; pp 384-391

Any inquiry concerning this communication or earlier communications from the Office should be directed to Melvin Bell whose telephone number is 571-272-3680. This Examiner can normally be reached on Mon - Fri 7:30 am - 4:00 pm.

If attempts to reach this Examiner by telephone are unsuccessful, his supervisor, Anthony Knight, can be reached on 571-272-3687. The fax phone number for the organization where this application or proceeding is assigned is (703) 872-9306.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 571-272-2100.

Art Unit: 2121

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

MB *MB*  
March 28, 2005

*Anthony Knight*  
**Anthony Knight**  
**Supervisory Patent Examiner**  
**Group 3600**

*If you need assistance in completing the form, call 1-800-PTO-9199 (1-800-786-9199) and select option 2.*

Substitute for form 1449A/PTO				APR 16 2002		Complete if Known	
<b>INFORMATION DISCLOSURE STATEMENT BY APPLICANT</b>  (use as many sheets as necessary)				Application Number	10/037,040		
				Filing Date	12/21/2001		
				First Named Inventor	Betty A. McDaniel		
				Group Art Unit	2183		
				Examiner Name	Unassigned		
Sheet	1	of	1	Attorney Docket Number	Baker 4-15-3-2-2		

[illegible][illegible]

Examiner Signature	<i>M. L. L.</i>	Date Considered	3/23/05
--------------------	-----------------	-----------------	---------



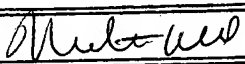
\* EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

<sup>1</sup>Unique citation designation number. <sup>2</sup>See attached Kinds of U.S. Patent Documents. <sup>3</sup>Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). <sup>4</sup>For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. <sup>5</sup>Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST. 16 if possible. <sup>6</sup>Applicant is to place a check mark here if English language Translation is attached.



# ELECTRONIC INFORMATION DISCLOSURE STATEMENT

Electronic Version v18  
Stylesheet Version v18.0

Title of Invention	Method of Improving the Lookup Performance of Tree-Type Knowledge Base Searches							
Application Number: 10/037040							<b>RECEIVED</b> SEP 05 2003 Technology Center 2100	
Confirmation Number: 5546								
First Named Applicant: Betty McDaniel								
Attorney Docket Number: 075903-091								
Art Unit: 2183								
Search string: ( 6266706 ).pn.								
<b>US Patent Documents</b>								
Note: Applicant is not required to submit a paper copy of cited US Patent Documents								
init	Cite.No.	Patent No.	Date	Patentee	Kind	Class	Subclass	
	1	6266706	2001-07-24	Brodnik, et al	B1	709	242	
<b>Signature</b>								
Examiner Name				Date				
				3/23/05				

<b>Notice of References Cited</b>	Application/Control No. 10/037,040	Applicant(s)/Patent Under Reexamination MCDANIEL ET AL.	
	Examiner Meltin Bell	Art Unit 2121	Page 1 of 2

**U.S. PATENT DOCUMENTS**

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-5,463,777	10-1995	Bialkowski et al.	707/102
	B	US-6,571,238	05-2003	Pollack et al.	707/5
	C	US-6,625,591	09-2003	Vahalia et al.	707/1
	D	US-6,636,802	10-2003	Nakano et al.	701/208
	E	US-6,662,184	12-2003	Friedberg, Stuart A.	707/100
	F	US-6,772,223	08-2004	Corl et al.	709/238
	G	US-6,516,319	02-2003	Benayoun et al.	707/100
	H	US-5,963,675	10-1999	van der Wal et al.	382/260
	I	US-6,260,044	07-2001	Nagral et al.	707/102
	J	US-6,563,952	05-2003	Srivastava et al.	382/225
	K	US-6,061,712	05-2000	Tzeng, Hong-Yi	709/202
	L	US-5,983,224	11-1999	Singh et al.	707/6
	M	US-4,611,272	09-1986	Lomet, David B.	707/3

**FOREIGN PATENT DOCUMENTS**

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

**NON-PATENT DOCUMENTS**

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	Demuyne et al; Bmad-tree: an efficient data structure for parallel processing; Eighth IEEE Symposium on Parallel and Distributed Processing; 23-26 Oct. 1996; pp 384-391
	V	
	W	
	X	

\*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)  
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

<b>Notice of References Cited</b>	Application/Control No. 10/037,040	Applicant(s)/Patent Under Reexamination MCDANIEL ET AL.	
	Examiner Meltin Bell	Art Unit 2121	Page 2 of 2

**U.S. PATENT DOCUMENTS**

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-6,381,607	04-2002	Wu et al.	707/100
	B	US-5,968,109	10-1999	Israni et al.	701/208
	C	US-5,404,513	04-1995	Powers et al.	707/102
	D	US-5,295,261	03-1994	Simonetti, Charles T.	707/2
	E	US-5,630,125	05-1997	Zellweger, Paul	707/103R
	F	US-5,930,805	07-1999	Marquis, Jean A.	707/201
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

**FOREIGN PATENT DOCUMENTS**

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

**NON-PATENT DOCUMENTS**

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

\*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)  
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.



# $B^{mad}$ -Tree: An Efficient Data Structure for Parallel Processing\*

Sajal K. Das  
Department of Computer Sciences  
University of North Texas  
P.O. Box 13886  
Denton, TX 76203-6886  
das@cs.unt.edu

Marie-Anne Demuynck  
Dept of Math and Computer Science  
Texas Woman's University  
P.O. Box 425589  
Denton, TX 87204-3589  
f.demuynck@venus.twu.edu

## Abstract

*B-trees are used for accessing large database files, stored in lexicographic order on the secondary storage devices. Algorithms for concurrent B-tree data structures achieve only limited speedup when implemented on a parallel computer. To improve the performance, we propose a variant of the  $B^{link}$ -tree, called the  $B^{mad}$ -tree, which allows insertion without node splits, with multiple access in its leaf nodes, and dilation in both the index and the leaf nodes. Parallel algorithms for search, insert and restructuring are designed for partitioned, locked and distributed models. Only part of an insertion node is locked during the insert, and simultaneous insertions by multiple processors in the same node are allowed. A restructuring algorithm runs periodically in the background and requires at most one wait by any search or update operation. Our implementations demonstrate that the  $B^{mad}$ -tree algorithms outperform the best known  $B^{link}$ -trees, and compare favorably with linear hashing. We achieve good speedup (e.g., 4.79 with 8 processors) for partitioned algorithms, and moderate speedup (2.49 with 8 processors) for locked algorithms, even including overhead costs. The insert times obtained for  $B^{mad}$ -trees are 50% to 60% less than that for the  $B^{link}$ -trees in partitioned implementations, and 70% to 80% less in locked implementations. The speedup results on the distributed memory platform (a network of workstations) were not that encouraging due to high communication costs.*

## 1. Introduction

B-trees are widely used as an access method for large ordered files stored on secondary storage devices [1], because they provide fast access and easy maintenance. A B-tree of order  $m$  has the following properties: (i)

every node has at most  $m$  and at least  $\lceil m/2 \rceil$  children, (ii) the root has at least two children, unless it is the only node, (iii) all leaf nodes appear at the same level of the tree, (iv) an internal node with  $k$  children, contains  $k - 1$  key values. This definition guarantees that a B-tree is at least half full at all times. There are several variants such as  $B^+$ -trees,  $B^{link}$ -trees and 2-3 trees [4, 13]. The  $B^+$ -tree is a B-tree in which each node is at least  $\frac{2}{3}$  full. In the  $B^{link}$ -tree, all keys reside in the leaves (the *sequence set*) and internal nodes (the *index set*) are used as an index to the sequence set. The 2-3 tree is a B-tree of order  $m = 3$ . For sequential algorithms on manipulating various B-trees, readers may refer to [1, 4, 13].

Several lock-based concurrent algorithms for the standard B-trees and  $B^+$ -trees are reported in [2, 11, 14, 25], and for the 2-3 trees in [7]. Mond and Raz [20] used *preparatory operations*, where possible node splits and merges are detected and executed during the search phase. Mohan [19] suggested locks on individual key values instead of entire nodes. Many lock-based concurrent algorithms also exist for the  $B^{link}$ -tree [5, 9, 15, 24], which is a  $B^+$ -tree variant where a pointer is added to each node linking it with its right neighbor. Other methods for achieving good performance include the use of cache memory [27], synchronization [10, 22], and retry strategies [17]. Distributed solutions due to Pramanik and Kim [23], and Matsliach and Shmueli [18], distribute the B-tree among the disks, require synchronized disk access mechanisms, and allow only a single operation at a time.

Although many concurrent algorithms have been proposed for B-trees and their variants, only a few deal with actual performance studies on real multiprocessors. Most of the existing experimental studies are simulations of concurrent B-trees on parallel architectures. Mukkamala and Shultz [21] measured the performance of the  $B^{link}$ -tree on two simulated architectures, one

\*This work is supported by Texas Advanced Technology Program under Award No. TATP-003594-31.

with shared output devices, and the other with one output device for each processor. Ford et al. [8] performed a simulation study of the concurrent B-trees due to [2] and [15] for a central file server application. The simulation study by Srinivasan and Carey [26] is for the same algorithms used in [8] and the B-tree algorithm due to [14]. Colbrook et al. [3] implemented their message passing algorithms on a simulator. Johnson and Shasha [12] proposed an analytical performance model for concurrent B-tree algorithms, and used simulation to substantiate their results.  $B^{link}$ -trees show the best results in all these studies.

Our earlier experiments [6] on concurrent  $B^{link}$ -trees on real parallel machines showed that a substantial amount of time was directly due to node splits. As the number of processors increases, the split time in  $B^{link}$ -trees may require more than 60% of insert time. We also observed a very large overhead in some algorithms due to locking. These results motivated us to develop a data structure in which splits are not necessary and restructuring is postponed, while nearly preserving sequential ordering of the data. Thus evolved the concept of the  $B^{mad}$ -tree, which stands for  *$B^{link}$ -tree with multiple access and dilation*. This new data structure is found to be more suited for efficient parallel processing. It allows insertion without node splits, access by multiple processors in its leaf nodes, and dilation in both the index and the leaf nodes.

In this paper, we design and implement algorithms for construction, search, insert, and restructure of  $B^{mad}$  for each of the three models - partitioned, locked, and distributed. The partitioned and locked algorithms are implemented on the Sequent Symmetry S/81 shared memory multiprocessor, and the distributed algorithms are implemented on a network of processors running the parallel virtual machine (PVM) software. Our experimental results compare favorably with those from  $B^{link}$ -tree and linear hashing implementations [6]. We achieve good speedup for partitioned algorithms (for example, a speedup of 4.79 with 8 processors), and moderate speedup for locked algorithms (speedup 2.49 with 8 processors), even including overhead costs. Efficiency is as high as 80% excluding overhead costs, and as high as 60% when overhead costs are included. The insert times obtained for  $B^{mad}$ -trees are 50% to 60% less than those for the  $B^{link}$ -tree in partitioned implementations, and 70% to 80% less in locked algorithms. The results on the distributed memory platform were not that encouraging, due to high startup and communication costs which prevented good speedup.

The paper is organized as follows. Section 2 introduces the  $B^{mad}$ -trees, and Section 3 describes parallel

algorithms for partitioned and locked models. Section 4 presents the implementation details and discusses the experimental results. Conclusions and future research are given in Section 5.

## 2. The $B^{mad}$ -tree Data Structure

The motivation behind the  $B^{mad}$ -trees originated from our experimental research on the concurrent  $B^{link}$ -trees and linear hashing on the Sequent Multiprocessor system [6]. The goal is to minimize very expensive lock and node split time, and allow multiple processors to access the same node during updates. The  $B^{mad}$ -tree does not require node split or merge, and restructuring is postponed, while nearly preserving data ordering.

Physically, a  $B^{mad}$ -tree is a  $B^{link}$ -tree where each leaf node is organized as a hash table with  $b$  buckets. The tree has the key values in semi-sorted order in the leaf level such that for any two leaf nodes,  $L_i$  and  $L_j$ , where  $L_i$  is to the left of  $L_j$ , all keys in  $L_i$  are smaller than those in  $L_j$ . In an initial version of this data structure, the leaf nodes were organized into buckets, similar to those used in bounded disorder files [16]. However, it is not optimal in terms of space utilization. An improved version of the  $B^{mad}$ -tree optimizes leaf node space as well. The available key storage space is organized as a single array of multiple linked lists, one for each bucket. The variable, *freelist*, heads the list of available space. Each node has a header record, which points to the top of each bucket's list and counts the keys in the bucket. The field, *totalcount*, counts the keys stored in the node. The field, *linkf*, is the link pointer to the next bucket node to the right. When a node is full, the next insert goes to an overflow node, linked to the current node via an overflow pointer, *ofptr*. Overflow nodes have exactly the same structure as any other leaf node, but the initial one is the primary node. A restructure algorithm restores the tree to a well-balanced form.

The modified leaf node structure eliminates wasted space. In this  $B^{mad}$ -tree, it does not matter if one bucket is large while another remains nearly empty. In the unlikely event that all the keys hash to the same bucket, performance is the same as any other  $B^+$ -tree. The use of many small buckets rather than a few larger buckets keeps search times short and allows greater concurrency, although this requires more overhead in the form of a larger header record.

## 3. Parallel Algorithms for the $B^{mad}$ -tree

This section contains an informal description of parallel  $B^{mad}$ -tree algorithms for construction, search, insert and restructure. For formal details, refer to [6].

### 3.1. Partitioned Algorithms

Most concurrent algorithms in the literature for B-trees use either locks or processor synchronization for concurrency control. In partitioned algorithms, introduced in [6], however, no locks are used and processors work independently from each other on a partition of the data structure.

The construction algorithm builds the tree from the bottom up, one leaf node at a time. The algorithm expects the keys to be in sorted order. New bucket nodes are generated and the index is constructed as needed, as long as there are keys in the file. Once the tree is built, the function *balance\_tree* checks the rightmost node at each level and shifts values from its left neighbor when an underfull node is found.

All searches start at the root node. As each index node is reached, its obsolete flag is checked. If it is set, the node is no longer valid and the search continues via the link pointer to the right neighbor. At the leaf node level, a search key is hashed to find its bucket. If the search key is not found and if there is an overflow node, the search resumes in the overflow node, until either the search key or the end of the chain is found. Since it is the task of the search algorithm to check the overflow chain and return the correct insertion location, an insert operation only adds new values to the node. No data are shifted, no nodes are split or merged, and no entries are added to the index. If there is room in the node, the new key is inserted at the top of the list in the target bucket. In the worst case the current bucket node is full, and a new overflow node is created before insertion takes place. The new node is then linked to the previous one by an overflow pointer.

The restructure algorithm is triggered when the tree has too many overflow nodes. This condition can be detected (i) either on a continuous basis as the tree accommodates update transactions, (ii) or by a separate function so that each processor keeps track of the number of overflow nodes it creates, and those counters are periodically checked by an idle processor. The first method works very well for sequential and partitioned algorithms, since there is no contention to update an overflow node counter. The second method is better for other types of concurrent algorithms. Levels are restructured one at a time, from left to right, by various rebuilding functions, one for the leaf nodes, one for the internal index nodes, and another for the root node.

A restructuring processor checks each leaf node and examines the parent node to ensure it is the correct one. If not, the link pointer is followed until the correct one is found. Each time a parent is first entered, it builds a new parent node, *ptemp*, and marks the old parent node obsolete. Each primary leaf node is examined in

turn. If it has no overflow chain, it remains as is, and its maxvalue and pointer are copied to *ptemp*. Otherwise, assume there are  $q > 1$  nodes in the chain. Keys in the primary and the overflow nodes are sorted and  $q$  new nodes are built and linked, the leftmost to the left neighbor and the rightmost one to the right neighbor. If there are enough new nodes, *ptemp* may fill up. A single index node then dilates into multiple nodes as shown in Figure 1. When a new parent node is started, the previous one is closed. If the last dilation node is underfull, enough key/pointer pairs are shifted from its left neighbor to keep the tree balanced. A leaf node rebuilding process of a chain with 3 nodes, and the corresponding changes in the deepest index level, are shown in Figure 2. Affected nodes are shown shaded in Figures 1 and 2, and all old pointers are shown dashed. An  $x$  is used to indicate an obsolete flag is set.

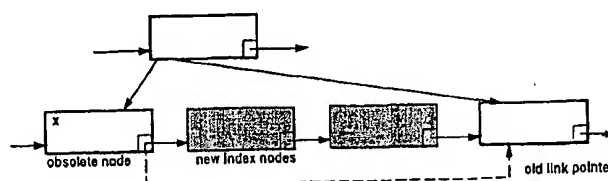


Figure 1. Temporarily expanded index node

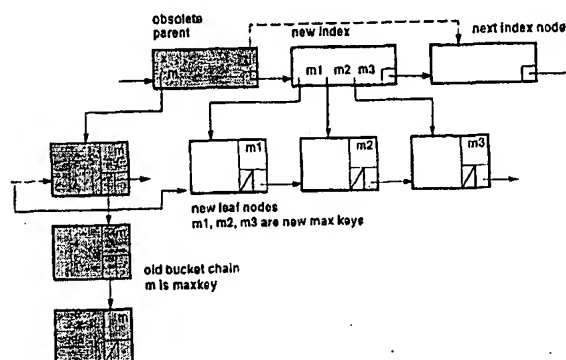


Figure 2. Restructuring of leaf nodes

Index levels are restructured next. Two levels are affected at a time, namely a lower level where obsolete nodes are removed, and its parent level where new nodes are built. In the former, nodes marked as obsolete are deleted and pointers to the new index nodes and their maxkeys are inserted in the new parent nodes. An index node with two dilations is shown in Figure 3. The root node is rebuilt as follows. If the level below the root node has fewer than  $m$  nodes, the old root is replaced by a single new one. If there are more than  $m$  nodes however, the tree expands and gets a new level and a new root node.

The restructure algorithm requires many duplicate nodes. In fact, an index level expands to at least twice its former size. The algorithm is improved by replacing obsolete parents immediately in those instances where there is only one dilation node. This saves space and shortens the time it takes to restructure the next level.

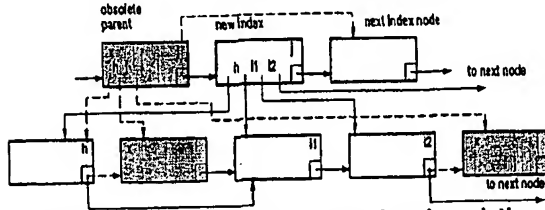


Figure 3. Newly rebuilt index level and the changes to its parent nodes

### 3.2. Parallel Algorithms with Locks

The search function for the locked algorithms proceeds as in the partitioned algorithms. Four cases arise: Case 1: No restructuring is in progress in the tree. Other cases occur due to the tree reconstruction. Case 2: an obsolete node who's replacement node(s) is(are) complete. Case 3: the replacement node(s) is(are) not finished. Case 4: a leaf node is restructuring.

If no restructuring is in progress, all checks for obsolete nodes are negative. Thus Case 1 covers the majority of all search operations. To prevent premature access to a not completely redone or to a deleted node, each index node has a flag, *linkflag*, which signals that the replacement list is complete. Each search process first checks a node's obsolete flag. If the flag is turned on, it checks *linkflag*. If it is set, the replacement nodes are ready to be used (Case 2) and the search continues. Otherwise the search waits until the list is ready (Case 3). In Case 4, the search process has reached the last index level and gives access to the target node. Delay is possible in at most one index node in the entire tree, because restructuring proceeds one level at a time. When an obsolete node is encountered, the search temporarily continues to the right instead of downwards through link pointers.

As the insert algorithm does not alter the index set of the tree, no locks are needed until the key is inserted in the node. There are three possibilities:

- (i) The insert node is not full. A lock on the bucket, its header and the head of the free list is obtained. The node is rechecked to prevent reinsertion of the same record. The node variables are updated, the lock is released, and the new record is added to the bucket.
- (ii) If the insert node is full, the overflow pointer is checked. If an overflow node already exists, the current

pointer is advanced, and insert completes as in (i).

- (iii) The current node is full but a new overflow node does not exist yet. A lock is requested on the overflow pointer. Once granted, the pointer is rechecked. If it is no longer *nil*, the lock is released and insert reverts to (ii). Otherwise, a new node is created and insertion completes. The new node is attached to the previous and the lock is released. The processor who creates a new overflow node increments its overflow counter.

In the locked restructure algorithm, an idle processor checks the restructure control variable, *restr\_flag*, which can have a value of 0, 1, or 2. If the flag is 0, no restructure is in progress, while 1 implies that restructure is currently in progress. A completed restructure is indicated by flag value 2. When flag is 0, the processor counts the total number of overflow nodes and starts restructuring if this number exceeds the limit. Initially the current processor activates the restructure driver, which controls two important events. First, only one processor at a time is allowed to restructure which is controlled by a busy flag. The second function of the driver is to know which is the next level to restructure. This is controlled by a variable *r.status*. At the leaf level, a node without overflow does not change and therefore need not be locked. The insertion of its key/pointer pair in the temporary parent node does not require locks either because only the restructuring processor has access to a dilation node at this time. When the replacement list is complete, the link flag in the old node is set. This also does not require a lock, since again only the restructuring processor is allowed to change that field and the obsolete flag. At the leaf level, nodes with chain lengths  $\geq 1$  are locked.

### 3.3. Speedup Analysis

Given a logical  $B^{mad}$ -tree of order  $m$  and of  $n$  key values, the total number of leaf nodes is  $v = \frac{n}{m}$ . The sequential time for single search/insert operation is  $T_s = O(\log_m v)$ . The performance of the partitioned  $B^{mad}$ -tree algorithms is obtained as follows. Each partition has approximately  $\frac{v}{p}$  leaf nodes, where  $p$  is the number of available processors. The height of the tree associated with each partition is  $O(\log_m \frac{v}{p})$ . Thus within a partition, a single update operation such as search and insert, requires  $T_p = O(\log_m \frac{v}{p})$  time. Therefore, the speedup for a single search and insert is  $S_p = \frac{T_s}{T_p} = O(\frac{\log_m v}{\log_m \frac{v}{p}})$ . The best case time for  $p$  simultaneous operations, where each processor or partition handles one operation, is also  $T_p = O(\log_m \frac{v}{p})$ , and speedup for  $p$  operations is  $S_p = O(\frac{p \log_m v}{\log_m \frac{v}{p}})$ . However, in the extreme worst case, all  $p$  operations may be performed in the same partition, and the parallel

time would be  $T_p = O(p \log_m \frac{v}{p})$ . By carefully pipelining, one may be able to reduce this time complexity. For the construction algorithm, sequential time is  $O(v \log_m v)$ . The tree creation time for each partition is  $T_p = O(\frac{v}{p} \log_m \frac{v}{p})$ , and hence the speedup is  $S_p = O(\frac{p \log_m v}{\log_m \frac{v}{p}})$ .

Let  $L_s$  be the time spent waiting for locks by the search process. Then the search time required in the lock-based algorithm is  $O(\log_m v + L_s)$ . If  $L_i$  represents the lock time spent by insert, then the traditional lock-based insert time is  $O(\log_m v + L_i)$ . The speedup for a single search operation is  $S_p = O(\frac{\log_m v}{\log_m v + L_s})$ . At most  $p$  operations take place at any one time, therefore  $S_p = O(\frac{p \log_m v}{\log_m v + L_s})$ . A similar analysis holds for insert.

#### 4. Performance Evaluation

The  $B^{mad}$ -tree algorithms were implemented for locked, partitioned and distributed approaches. We focus on the search and insert operations, as well as on the construction and restructure. As expected, the  $B^{mad}$ -trees showed better performance than other  $B^{link}$ -trees, especially for locked implementations.

##### 4.1. Implementation Environments

The partitioned and locked algorithms were implemented on a Sequent Symmetry S/81 shared memory system with 16 processors, running under DYNIX O.S., a version of Unix 4.2bsd. The Sequent is a tightly coupled multiprocessor with 32-bit microprocessors and a shared bus. All the processors are identical. The programs were written in parallel C language. Measurements were taken using the Sequent's system clock, and are given in microseconds.

Distributed algorithms were implemented on a network that included DEC Alpha workstations and Intel 486 based PCs, running the Parallel Virtual Machine (PVM) version 3.3 software package. This software allows a network of heterogeneous computers to run as a single multicomputer system, called a *virtual machine*. It is a user defined collection of systems that can include serial, parallel and vector computers – all machines run under the UNIX. Once the virtual machine is defined, PVM includes routines to automatically spawn tasks on the member computers, and allows them to communicate and synchronize with each other.

##### 4.2. Performance Measurements

Relatively small trees are used as test data due to limited memory space available in our system. The  $B^{mad}$ -trees considered have index nodes of size  $m = 8$ ,

and leaf nodes of size = 20. The initial trees were constructed with full index nodes. Unlike other B-trees, it is efficient to construct the  $B^{mad}$ -tree with full index nodes, since subsequent inserts have no immediate effect on the index. This yields a denser, shorter (in height), and therefore a more efficient tree. The following time measurements were taken as appropriate for each algorithm.

- *Total run time ( $T_{total}$ )*: It measures the total time to process all the data in the input stream. It includes  $T_{insert}$ ,  $T_{oh}$ , and  $T_{fork}$ .  $T_{lock}$  is included for lock-based algorithms, and  $T_{comm}$  for distributed algorithms.
- *Insert time ( $T_{insert}$ )*: It measures the time for insertion, including the times  $T_{lock}$  and  $T_{ofcreate}$ .
- *Overhead time ( $T_{oh}$ )*: It measures the time needed to get the data ready, and the time required to access the input stream.
- *Process Creation time ( $T_{fork}$ )*: It measures only the time needed to spawn the parallel processes.
- *Lock time ( $T_{lock}$ )*: It measures the time spent obtaining and releasing locks. The system functions to obtain and release locks are *s.lock()* and *s.unlock()*. It does not include the amount of time the lock is held while updating locked data.
- *Send time ( $T_{send}$ )*: This is the time required to send messages in a distributed environment.
- *Receive time ( $T_{recv}$ )*: It measures the time waiting for messages to arrive.
- *Communication time ( $T_{comm}$ )*: This measures the total time spent in communication overhead. It is likely that  $T_{send}$  and  $T_{recv}$  overlap at least partially. We could not, however, accurately measure the amount of overlap. Therefore,  $T_{comm} = \max\{T_{send}, T_{recv}\}$ .
- *Overflow creation time ( $T_{ofcreate}$ )*: This measures the time to create new overflow nodes during insert operations.
- *Restructure time ( $T_{restr}$ )*: This measures the time required to restructure the tree. Note that this time is a background time and almost never interferes with other tree operations.

##### 4.3. Results for Search and Insert

The performance of the partitioned algorithms for  $B^{mad}$ -trees is good, and gradually improves as the number of transactions grows. The best results were observed at 10,000 insertions, even though the tree has now tripled in size. Figure 4 shows the components of the total insert time,  $T_{total}$ , for an initial tree of 5,000 keys with 10,000 subsequent inserts. This includes overhead time, fork time, search time and insert time. Both the pure insert and search operations show

steady increase in the speedup as the number of processors ( $p$ ) increases, with the higher speedup observed for the search operation. The speedup saturates beyond  $p = 10$ , due to the linear growth of  $T_{fork}$  with  $p$ . However, the overhead time,  $T_{oh}$ , remains constant. Figure 5 shows a typical result for a smaller number of transactions. There is good speedup through  $p = 6$ , when the linear increase in  $T_{fork}$  causes a decline in performance.

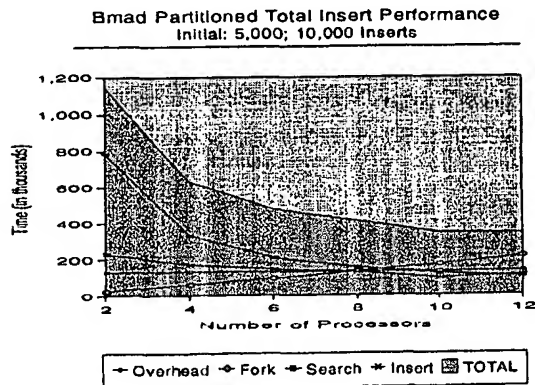


Figure 4.  $T_{total}$  (in microseconds) for  $B^{mad}$ -tree with 10,000 inserts

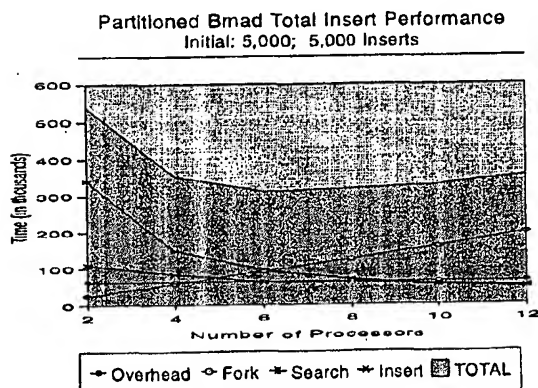


Figure 5.  $T_{total}$  (in microseconds) for  $B^{mad}$ -tree with 5,000 inserts

One should not think that the insertion in the  $B^{mad}$ -tree yields a better performance at the cost of increased search time, especially as the tree grows and the number of overflow nodes increases. The average search time per key, computed as  $\frac{T_{search}}{n/p}$ , shows only a slight increase in the search time. The average insert time per key,  $\frac{T_{insert}}{n/p}$ , decreases as  $n$  grows larger. Thus the average total time per key,  $\frac{T_{total}}{n/p}$ , decreases or remains constant. This trend is persistent, regardless of the number of processors in the system. A typical result is

shown in Figure 6.

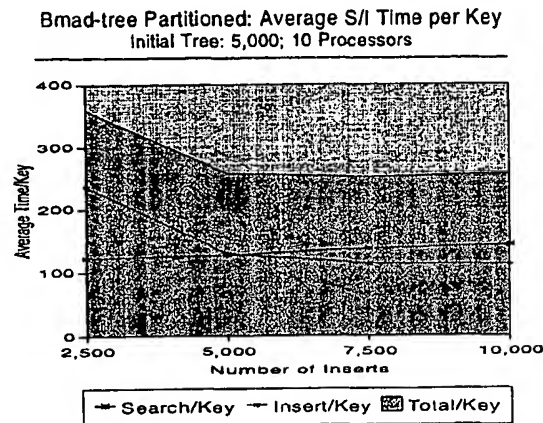


Figure 6. Average time per key in  $B^{mad}$ -tree

In the implementations for the distributed algorithms on a network of processors created with PVM, we were able to spawn many more processors than on the shared memory system. However, at about 18 to 20 processors, the results lost much of their significance because partitions become very small. We present results for up to  $p = 22$  in which the dominant features are the spawn and communication costs. These results are consistent for all combinations of initial trees and subsequent transactions. Figure 7 shows speedup performance. It includes costs for total insert, spawning the processes and communication. When evaluating these results however, we should remember that the process creation costs are a one-time cost in nearly all instances. Once the initial virtual machine is created, that cost will not reoccur unless a processor fails. Similarly, if communication costs are concentrated at the startup, they diminish proportionately as the processing time and the number of transactions increases.

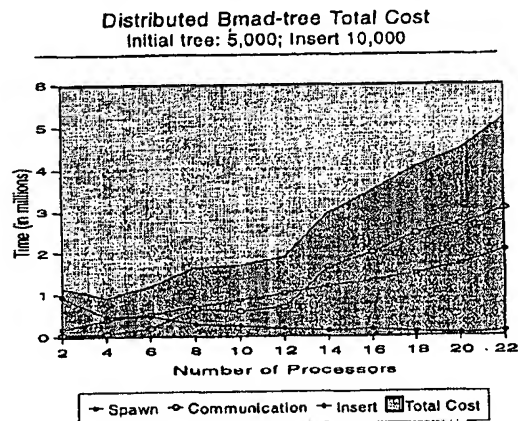


Figure 7.  $T_{total}$  for distributed  $B^{mad}$ -tree



Figure 8 shows the results for the locked  $B^{mad}$ -trees, which show good speedup initially but subsequent loss of performance due to the linear increase in fork time. We expected that the altered leaf node structure of the  $B^{mad}$ -tree and the elimination of index adjustments during insertions would reduce  $T_{lock}$  costs, but results were even better than expected, and  $T_{lock}$  time is consistently the smallest component within the insert time.

Table 1 gives speedup information for the various  $B^{mad}$ -tree algorithms, where P, L and D respectively denote 'partitioned', 'locked', and 'distributed'. Efficiency is defined as the ratio of speedup to  $p$ , and processor utilization is optimal when efficiency is  $O(1)$ . In our experiments, the partitioned algorithms achieve the highest efficiency, up to 0.81 for  $p = 8$  when no overhead is included, and 0.60 when overhead is included. The efficiency for the locked algorithms with  $p = 8$  is up to 0.48 (no overhead) and 0.31 (with overhead). For the distributed algorithms with  $p = 8$ , efficiency is as high as 0.85 (no overhead) and as low as 0.01 (with overhead).

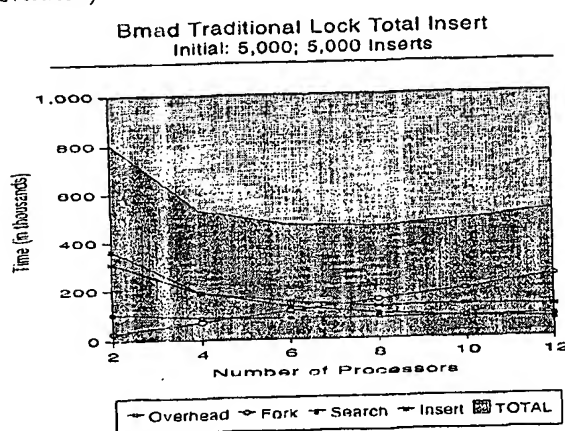


Figure 8.  $T_{total}$  for locked  $B^{mad}$ -tree

#### 4.4. Performance Results for Restructure

As expected, restructuring the tree is expensive. In partitioned and distributed algorithms,  $T_{restr}$  shows good speedup. At  $p = 2$   $T_{restr}$  is as much as  $T_{total}$ , but good speedup reduces that cost significantly as  $p$  increases. Similar but slightly better results were obtained in the distributed algorithms.

In the locked solutions, the results are viewed in two different ways. Like others, the restructure costs are measured sequentially. Since for locked algorithms restructuring happens on the entire tree, not on a partition, the cost is not distributed over all the processors. Thus  $T_{restr}$  is large and remains essentially the same, independent of the number of processors. This is

Table 1. Speedup for the  $B^{mad}$ -tree

Number of Transactions	No Overhead			With Overhead		
	P	L	D	P	L	D
8 processors						
5,000 keys	6.48	3.79	6.76	4.69	2.00	0.66
10,000 keys	6.57	3.85	6.94	4.79	2.49	1.22
12 processors						
5,000 keys	9.04	4.43	10.97	5.79	1.74	0.50
10,000 keys	8.68	4.83	9.19	5.75	2.53	1.07

clearly a false result, since the restructure runs simultaneously with other processing, or in the background.

#### 4.5. Comparison With Others

Figure 9 compares the total performance ( $T_{total}$ ) of the  $B^{mad}$ -tree with other concurrent data structures, including  $B^{link}$ -trees and linear hashing. The locked  $B^{link}$ -tree remains the most expensive solution, but distributed solutions are also expensive as the number of processors in the virtual machine grows. The locked  $B^{mad}$ -tree and linear hashing and all the partitioned algorithms give the best results. The insert times obtained for  $B^{mad}$ -tree are 50% to 60% less than those for the  $B^{link}$ -tree in partitioned implementations. The results are even better for locked algorithms, where the  $B^{mad}$ -tree implementations run in 70% to 80% less time than those for the locked  $B^{link}$ -tree algorithms. Figure 10 gives the results excluding overhead costs. It shows continuing speedup for all algorithms, except for locked linear hashing. Note that the data for lock-based  $B^{link}$ -trees are not included, as they are an order of magnitude higher than the other results obtained.

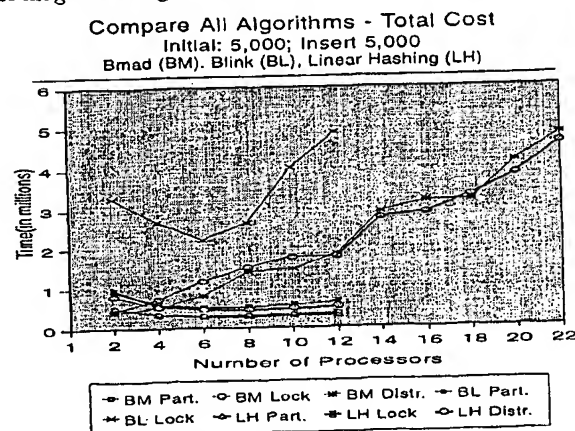


Figure 9. Comparing  $T_{total}$  for all algorithms

#### 5. Conclusions and Future Research

This research started with a simple premise. Many parallel and concurrent algorithms have been proposed for B-trees, but very few ever were implemented on real

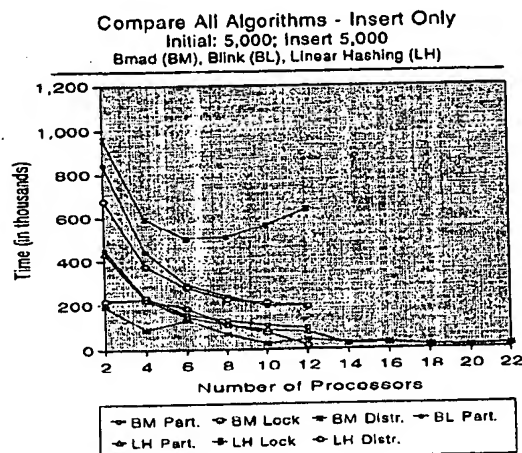


Figure 10.  $T_{insert}$  for all algorithms

machines. Most of those were tested on simulated multiprocessors and hence very limited data on the comparative performance of concurrent B-tree and hashing algorithms are available.

An improved *Blink*-tree, called the *B<sup>mad</sup>*-tree, is proposed in this paper. It allows multiple updates in the same node, reduces the number of locks required during updates and the length of time locks are held, and delays tree restructuring. The implementation results show that the *B<sup>mad</sup>*-tree algorithms perform better compared to other existing B-tree implementations. The locked *B<sup>mad</sup>*-tree algorithms require only 20% to 30% of the time required by the *Blink*-trees, and 40% to 50% of the time by the partitioned *Blink*-trees. The *B<sup>mad</sup>*-trees even outperformed the linear hashing, a fast hash table algorithm. The high cost of the restructure algorithm does not affect the overall performance, as it is intended to run in the background. Further investigations in network and communications aspects are needed to determine how to achieve better performance in the distributed algorithms.

Our future research includes the development of other algorithms for the *B<sup>mad</sup>*-trees, such as traversal, range search and non-trivial delete. The load balancing issues in the proposed parallel algorithms will also be studied by experiments.

## References

- [1] Bayer, R., and McCreight, C. "Organization and maintenance of large ordered indexes." *Acta Informatica*, 1 (1972) 173-189.
- [2] Bayer, R., and Schkolnick, M. "Concurrency of Operations on B-Trees." *Acta Informatica*, 9 (1977) 1-21.
- [3] Colbrook, A., Brewer, E.A., Dellaroccas, C.N., and Weihl, W.E. "Algorithms for Search Trees on Message-Passing Architectures." *IEEE Transactions on Parallel and Distributed Systems*, 7(2) (1996) 97-108.
- [4] Comer, D. "The Ubiquitous B-tree." *Computing Surveys*, 11, 2 (June 1979) 121-137.
- [5] de Jonge, W., and Schijf, A. "Concurrent Access to B-Trees." *Proc Int Conf on Databases, Parallel Architectures and their Applications* (Mar. 1990) 312-320.
- [6] Demuynck, M.-A., *Performance Study of Concurrent Search Trees and Hash Algorithms on Multiprocessor Systems.* Ph.D. Thesis, Univ. North Texas, Denton, 1996.
- [7] Ellis, C.S. "Concurrent Search and Insertion in 2-3 Trees." *Acta Informatica*, 14 (1980) 63-86.
- [8] Ford, R., Jipping, M., Shultz, R., and Wenhardt, B. "On the Performance of Concurrent Tree Algorithms." *Journal of Parallel and Distributed Computing*, 8 (1990) 253-266.
- [9] Fu, A., and Kameda, T. "Concurrency Control of Nested Transactions Accessing B-Trees." *Proc 8th ACM SIGACT-SIGMOD-SIGART Symp on Principles of Database Systems* (Mar. 1989) 270-285.
- [10] Higham, L., and Schenk, E. *Maintaining B-Trees on an EREW PRAM*. University of Calgary, Department of Computer Science, RR 91/446/30, 1992.
- [11] Ishak, R. "Semantically Consistent Schedules for Efficient and Concurrent B-tree restructuring." *Proc. 8th International Conference on Data Engineering* (1992) 184-191.
- [12] Johnson, T., and Shasha, D. "The Performance of Concurrent B-Tree Algorithms." *ACM Transactions on Database Systems*, 18 (1) (1993) 51-101.
- [13] Knuth, D.E. *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addison-Wesley, 1973.
- [14] Kwong Y.-S., and Wood, D. "A New Method for Concurrency in B-Trees." *IEEE Transactions on Software Engineering*, SE-8, 3 (May 1982) 211-222.
- [15] Lehman, P.L., and Yao, S.B. "Efficient Locking for Concurrent Operations on B-Trees." *ACM Transaction on Database Systems*, 6, 4 (1981) 650-670.
- [16] Lomet, D. "A Simple Bounded Disorder File Organization with Good Performance." *ACM Transactions Database Systems* 13(4) (1988) 525-551.
- [17] Loyd, L.D. *Concurrent Access to O-trees*. Thesis, Dept of Comput Science, Univ of Virginia, 1991.
- [18] Matsliach, G., and Shmueli, O. "Distributing a B+-Tree in a Loosely Coupled Environment." *Information Processing Letters*, 34 (1990) 313-321.
- [19] Mohan, C. "ARIES/KVL: A Key-Value Locking Method for Concurrency Control of Multiaccess Transactions Operating on B-tree indexes." *Proceedings of the 16th Conference on Very Large Data Bases* (1990) 392-405.
- [20] Mond, Y., and Raz, Y. "Concurrency Control in B+-Trees using Preparatory Operations." *Proc 11th Int Conf on Very Large Data Bases*, (Aug. 1985) 331-334.
- [21] Mulkamala, R., and Shultz, R.K. "Performance Comparison of two Multiprocessor B-link Tree Implementations." *Proc Int Conf Parallel Process*, Vol 1 (1988) 182-186.
- [22] Paul, W., Wishkin, U., and Wagener, H. "Parallel Dictionaries on 2-3 Trees." *Lecture Notes in Computer Science*, 154 (1983) 597-609.
- [23] Pramanik, S., and Kim, M.H. "Parallel Processing of Large Node B-trees." *IEEE Transactions on Computers*, 39, 3 (1990) 1208-1212.
- [24] Sagiv, Y. "Concurrent Operations on B+-Trees with Over-taking." *Journal of Computer and System Sciences*, 33 (1986) 275-296.
- [25] Samadi, B. "B-Trees in a System with Multiple Users." *Information Processing Letters*, 5, 4 (Oct., 1976) 107-112.
- [26] Srinivasan, V., and Carey, M.J. "Performance of B-tree Concurrency Control Algorithms." *Proc ACM SIGMOD Int Conf on Management of Data* (1991) 416-425.
- [27] Weihl, W., and Wang, P., "Multi-Version Memory: Software Cache Management for Concurrent B-Trees." *Proc Symp Parallel and Distributed Processing*, (1990) 650-655.



TC2100

Bldg./Room



02 1A  
0004204034  
MAILED FROM

Organization

U. S. DEPARTMENT OF COMMERCE  
COMMISSIONER FOR PATENTS

P.O. BOX 1450

ALEXANDRIA, VA 22313-1450

IF UNDELIVERABLE RETURN IN TEN DAYS

OFFICIAL BUSINESS

AN EQUAL OPPORTUNITY EMPLOYER

ATTEMPTED  
NOT KNOWN



RECEIVED  
APR 11 2005  
USPTO MAIL CENTER

RECEIVED  
APR 12 2005  
Technology Center 2100